Adaptive Placement for In-memory Storage Functions

Ankit Bhardwaj, Chinmay Kulkarni, and Ryan Stutsman

University of Utah

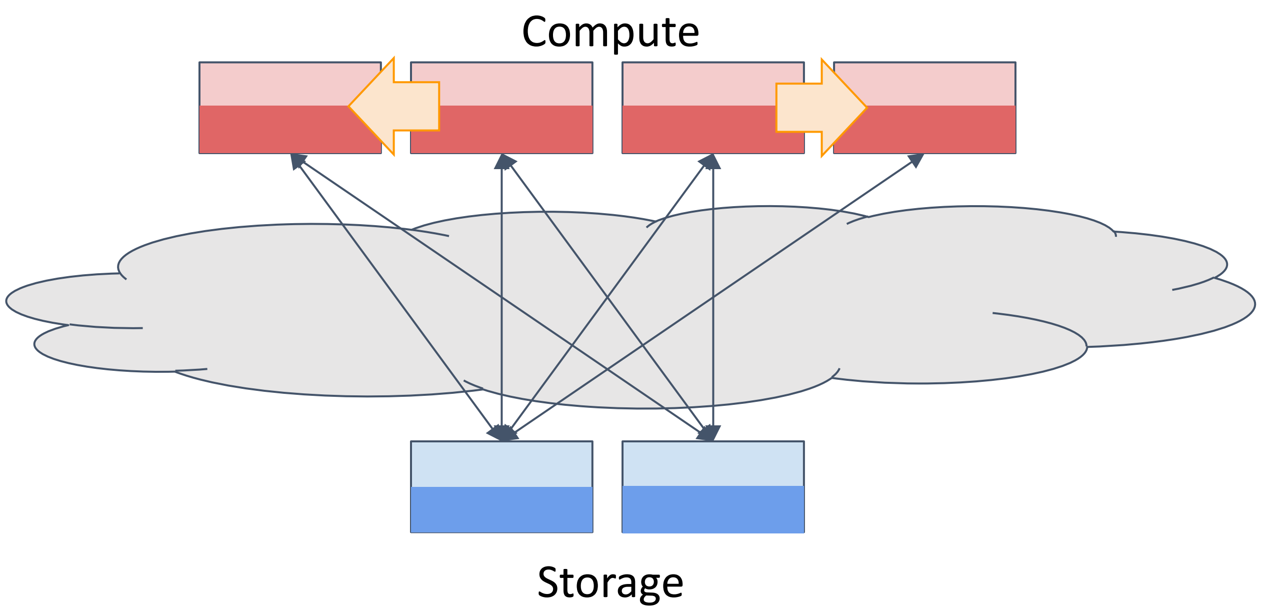
THE UNIVERSITY OF UTAH®

Utah Scalable Computer Systems Lab

Introduction

- **Kernel-bypass** key-value stores offer < **10μs** latency, **Mops** throughput
  - Fast because they are just dumb
  - Inefficient – Data movement, client stalls
- Run application logic on the server?
  - Storage server can become bottleneck, effects propagates back to clients
- **Key-ideas:** Put application logic in decoupled functions
  - Profile invocations & adaptively place to avoid bottlenecks
  - Challenge: efficiently shifting compute at microsecond-timescales

Disaggregation Improves Utilization and Scaling



Decouple Compute & Storage using Network

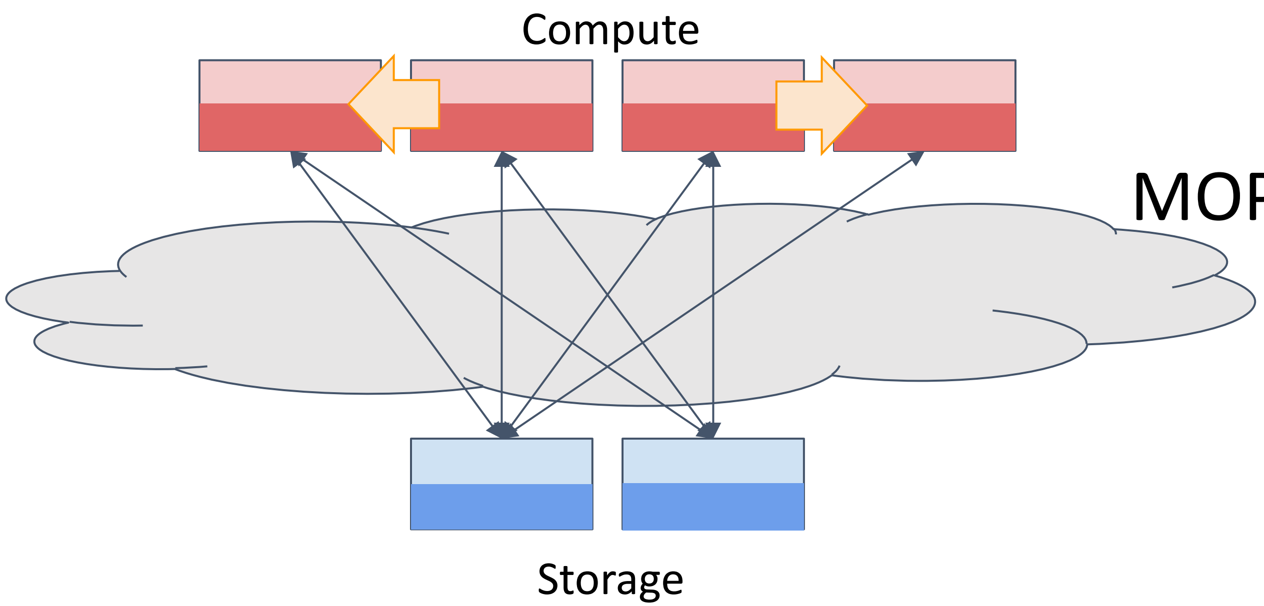
Provision at idle Capacity

Scale Independently

Disaggregation Improves Utilization and Scaling

FaRM

RAMCloud



<10µs latency

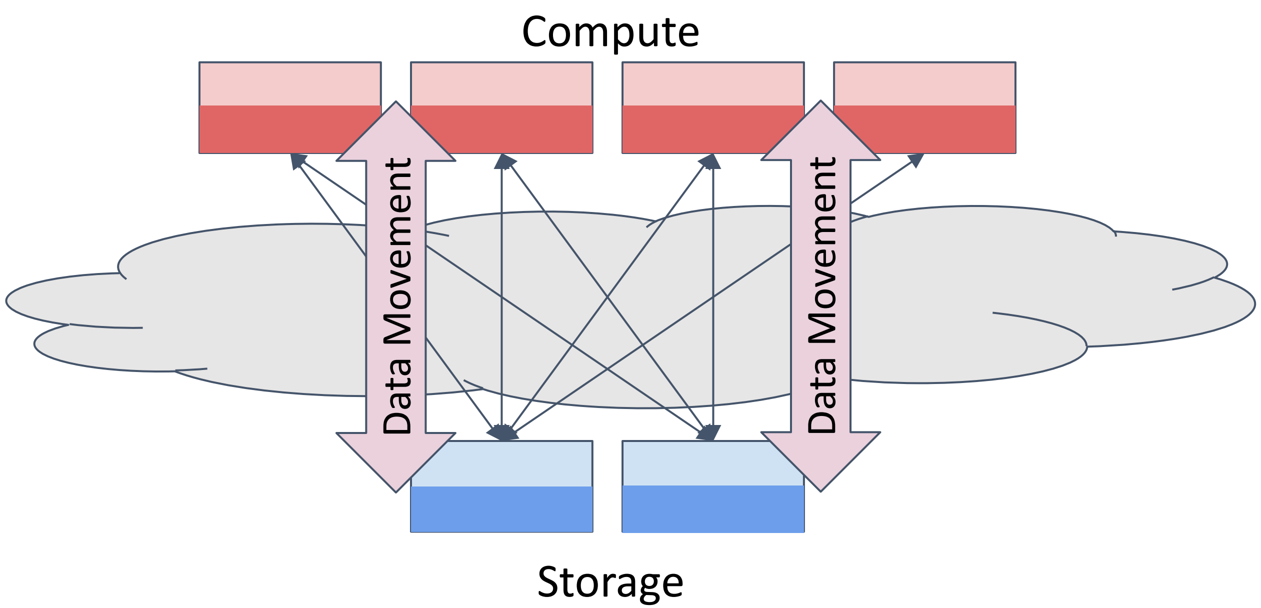
MOPS Throughput

Decouple Compute & Storage using Network

Provision at idle Capacity

Scale Independently

But, Data Movement Has a Cost



Massive Data Movement Destroys Efficiency

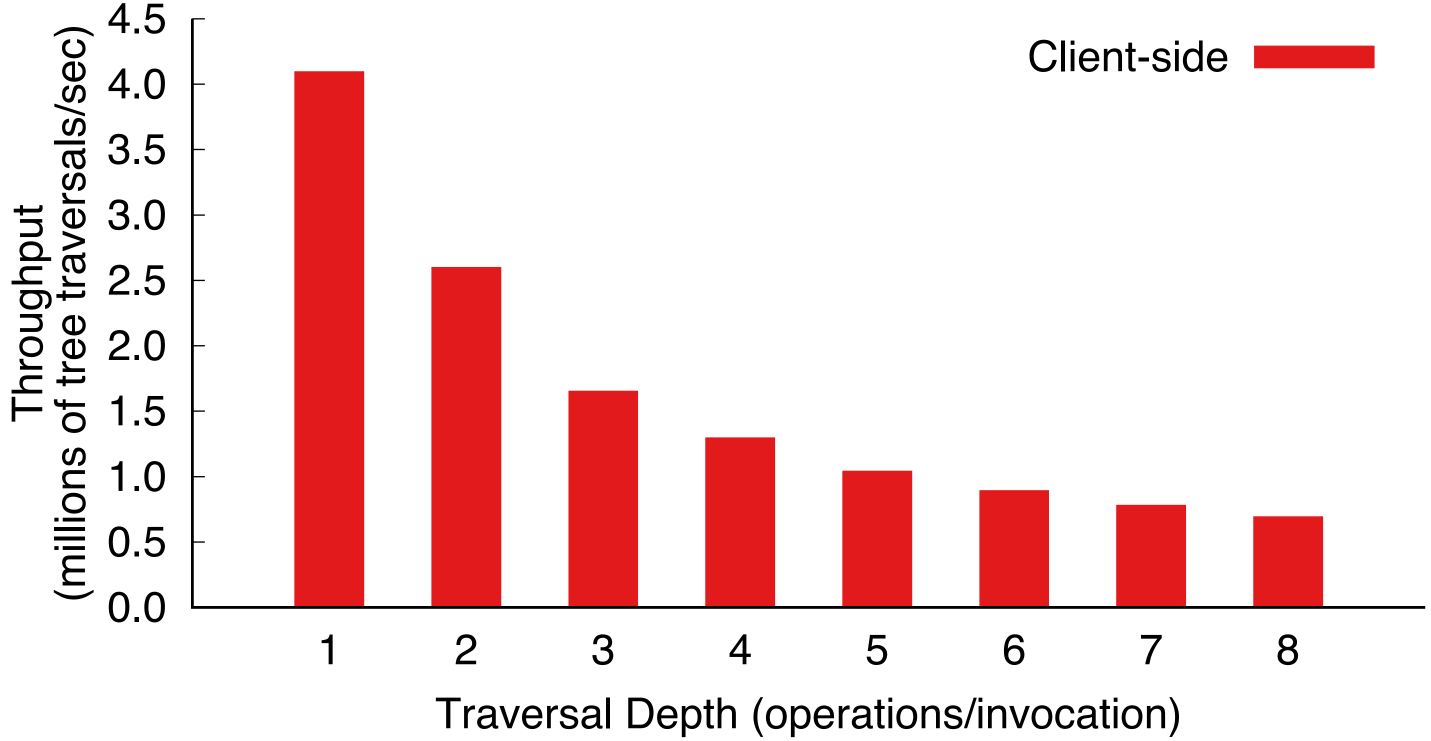
So, push code to storage?

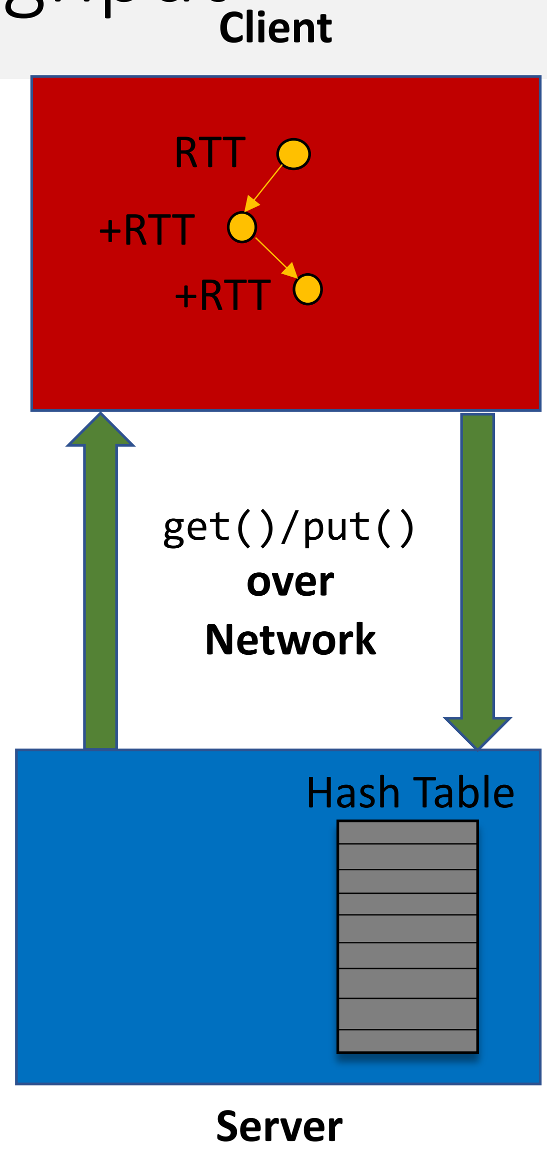
Storage Function Requirements

- Microsecond-scale -> low invocation cost
- High-throughput, in-memory -> native code performance
- Amenable to multi-core processing

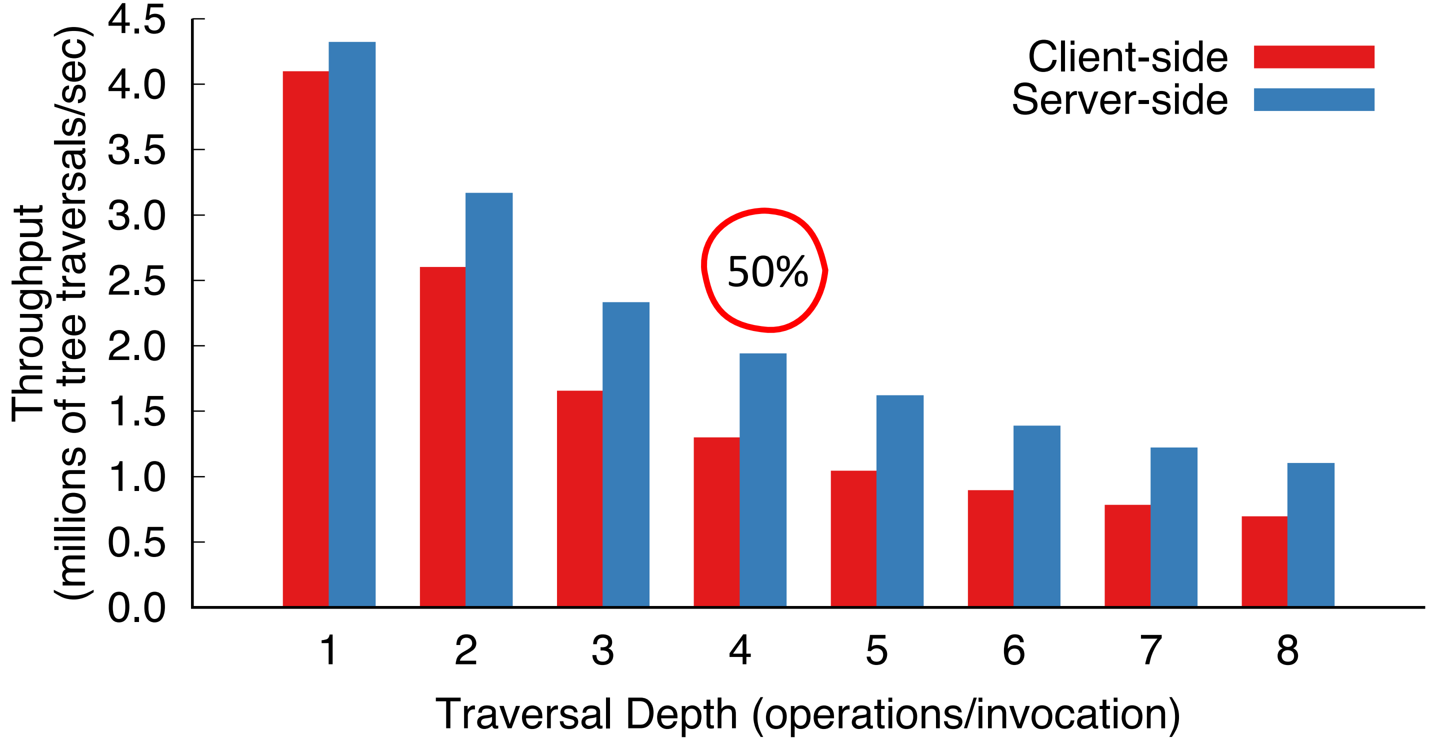
- Solution: Splinter allows loadable compiled extensions of storage functions

Server-side Placement Can Improve Throughput

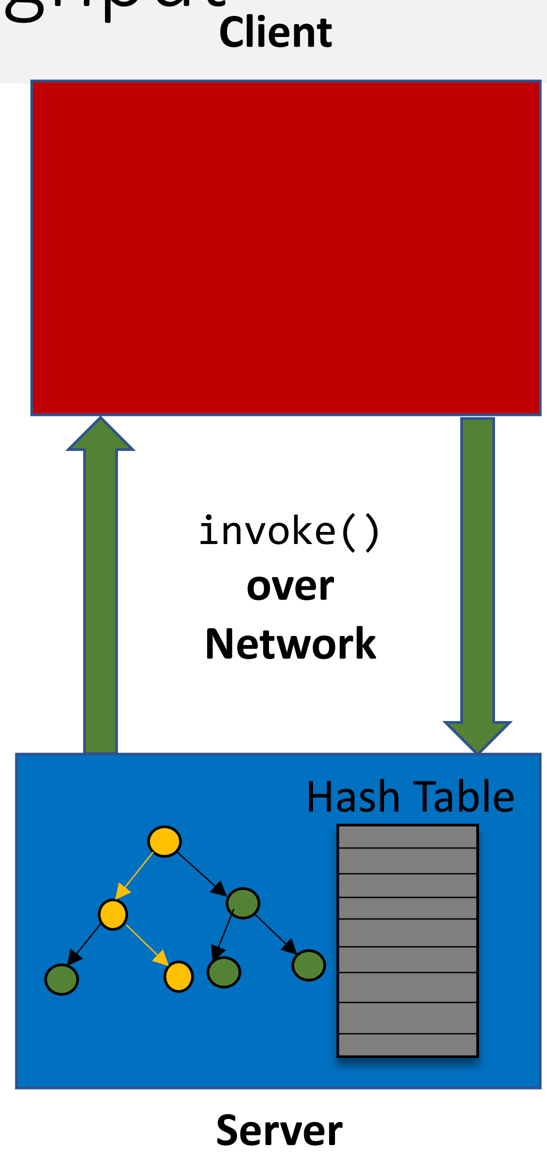




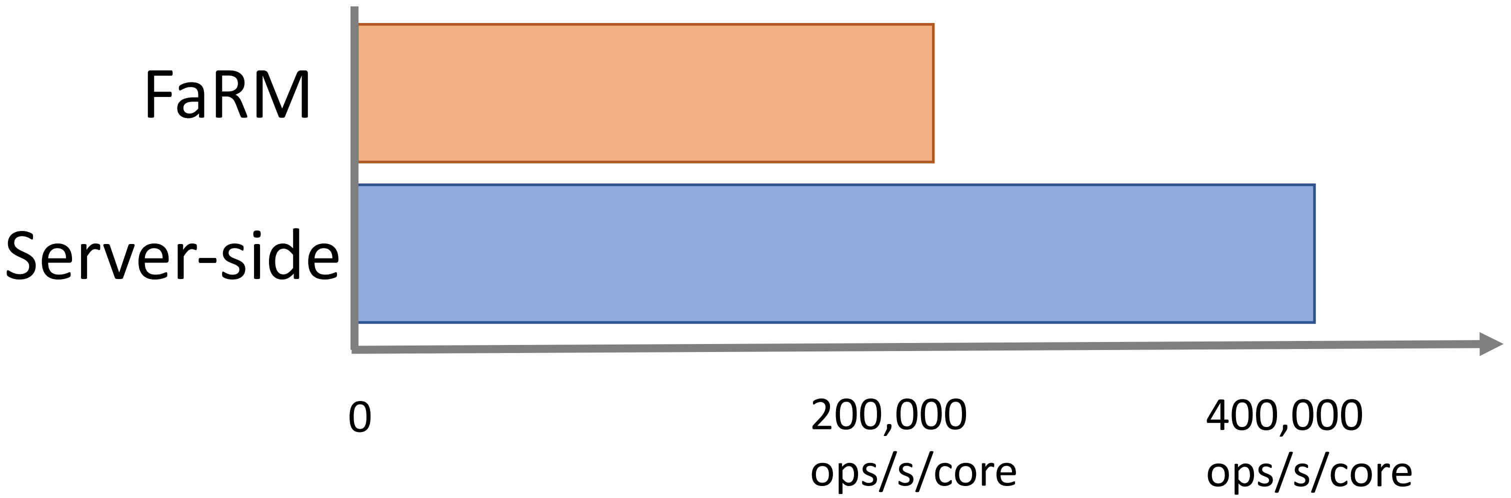
Server-side Placement Can Improve Throughput

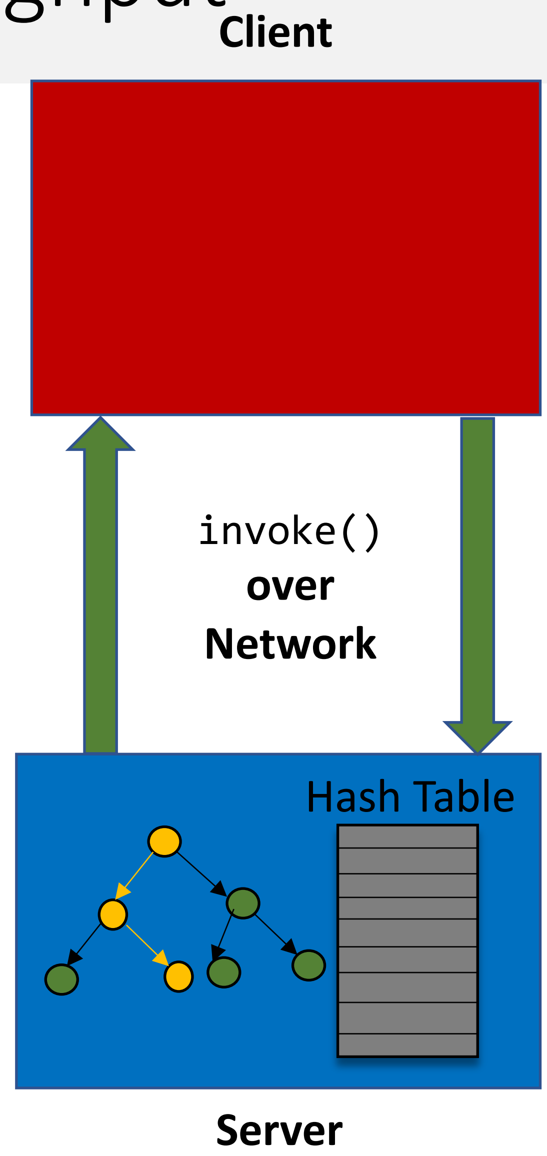


Reduces (N-1) RPCs and RTTs



Server-side Placement Can Improve Throughput





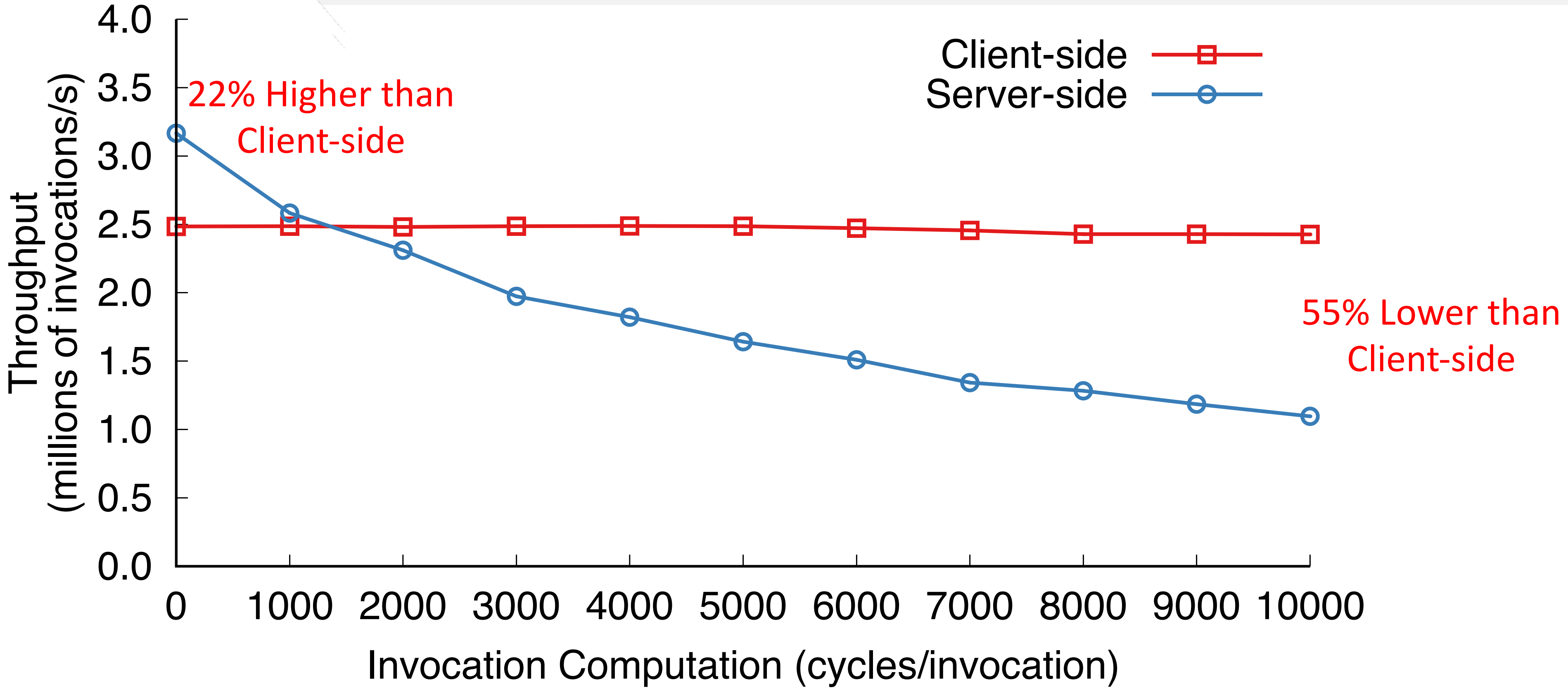
Facebook TAO graph operations perform 2x better as compared to state-of-the-art system FaRM

Server-side Placement Can Bottleneck the Server

- Server-side placement is good for data-intensive functions
- Compute-intensive functions make the server CPU bottleneck
- Overloaded server stops responding to even *get()/put()* requests

- Overall system throughput drops

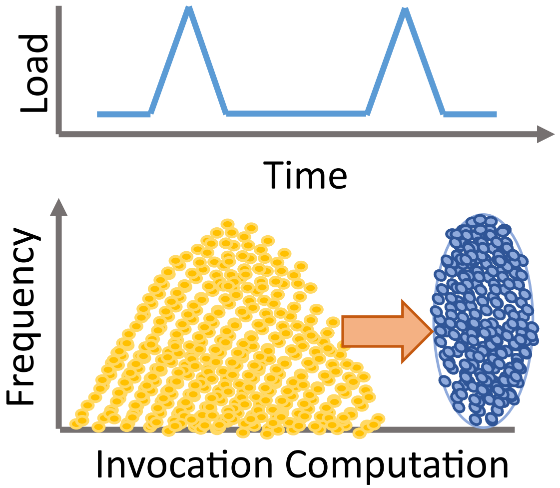
Server-side Placement Can Bottleneck the Server



Tree Depth 2

What about Rebalancing and Load-Balancing?

- Workload change can happen in two ways
  - Workload shifts in function call distribution over time
  - Shifts in per-invocation costs
- Migrate data only when the workload is stable

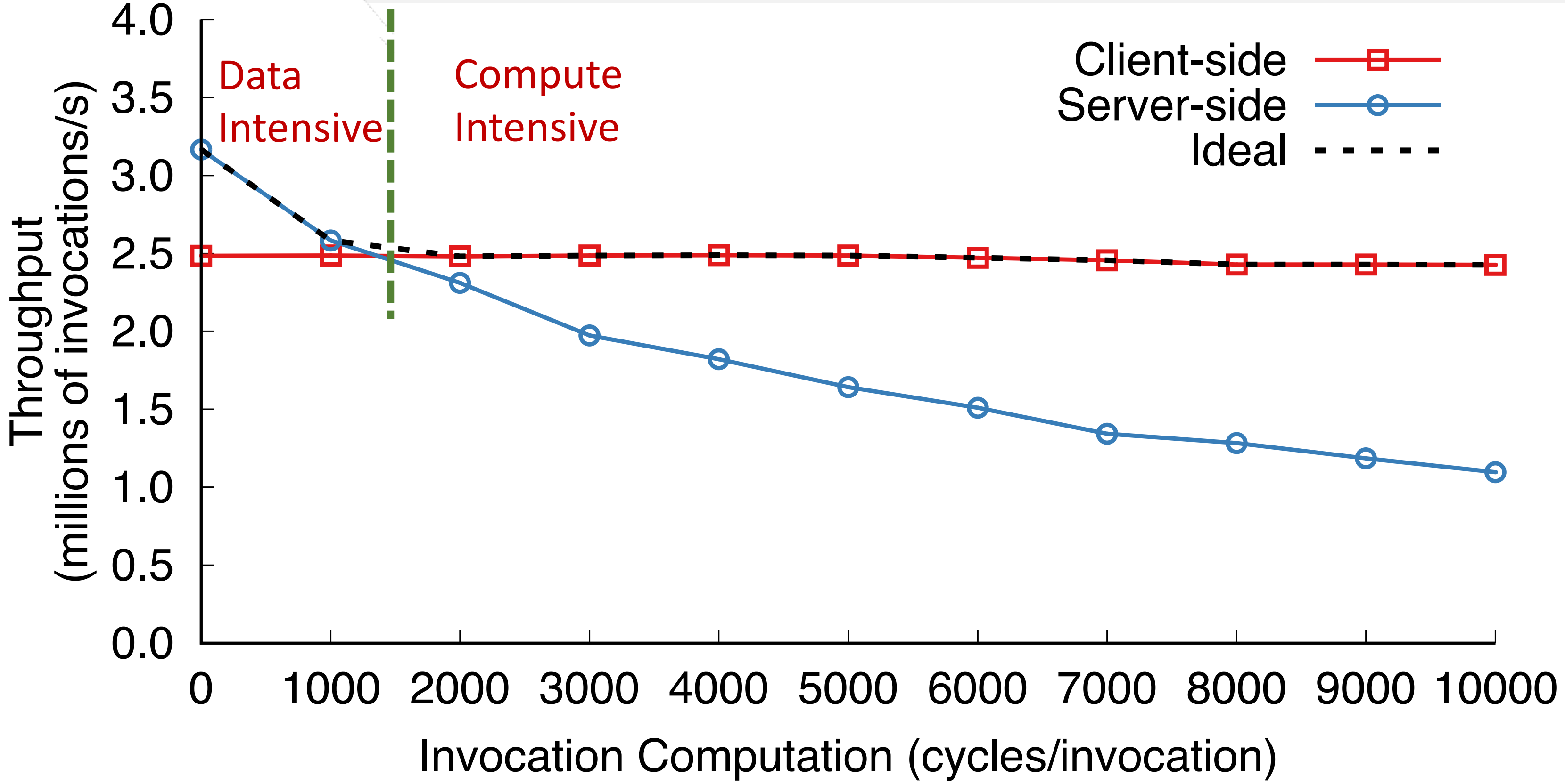


- Moving load to client and use the server CPU for migration

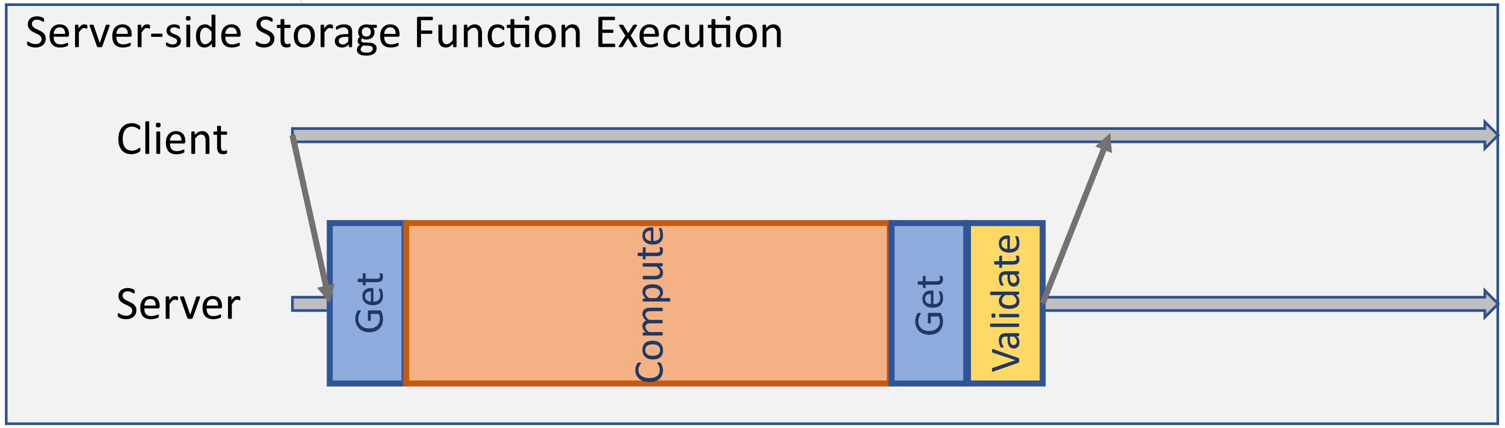
Key Insight: Decoupled Functions Can Run Anywhere

- Tenants write *logically* decoupled functions using standard get/put interface
- Clients *physically* push and run functions server-side
- Or the clients could run the functions locally

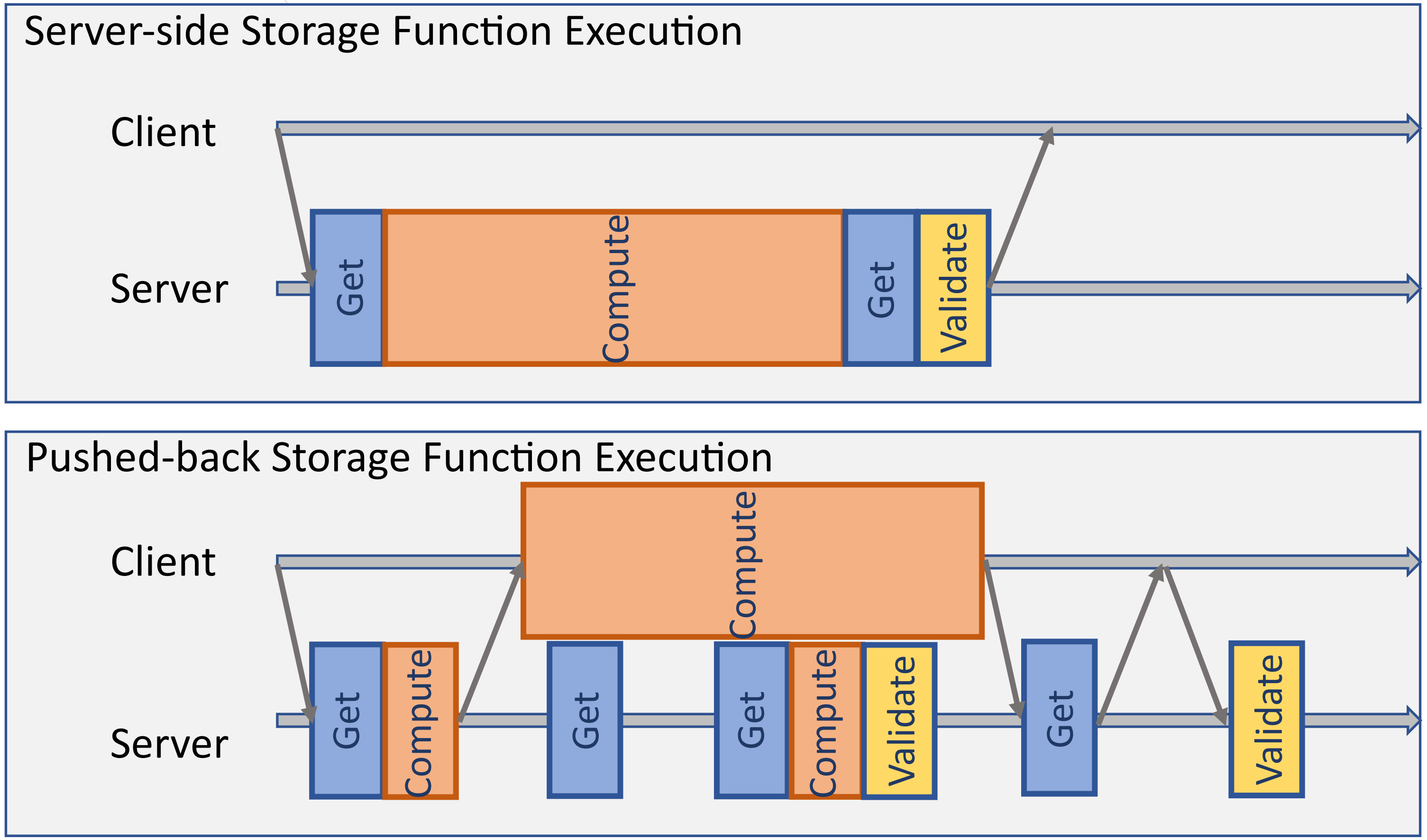
Goal: The Best of Both Worlds



Adaptive Storage Function Placement (ASFP)



Adaptive Storage Function Placement (ASFP)

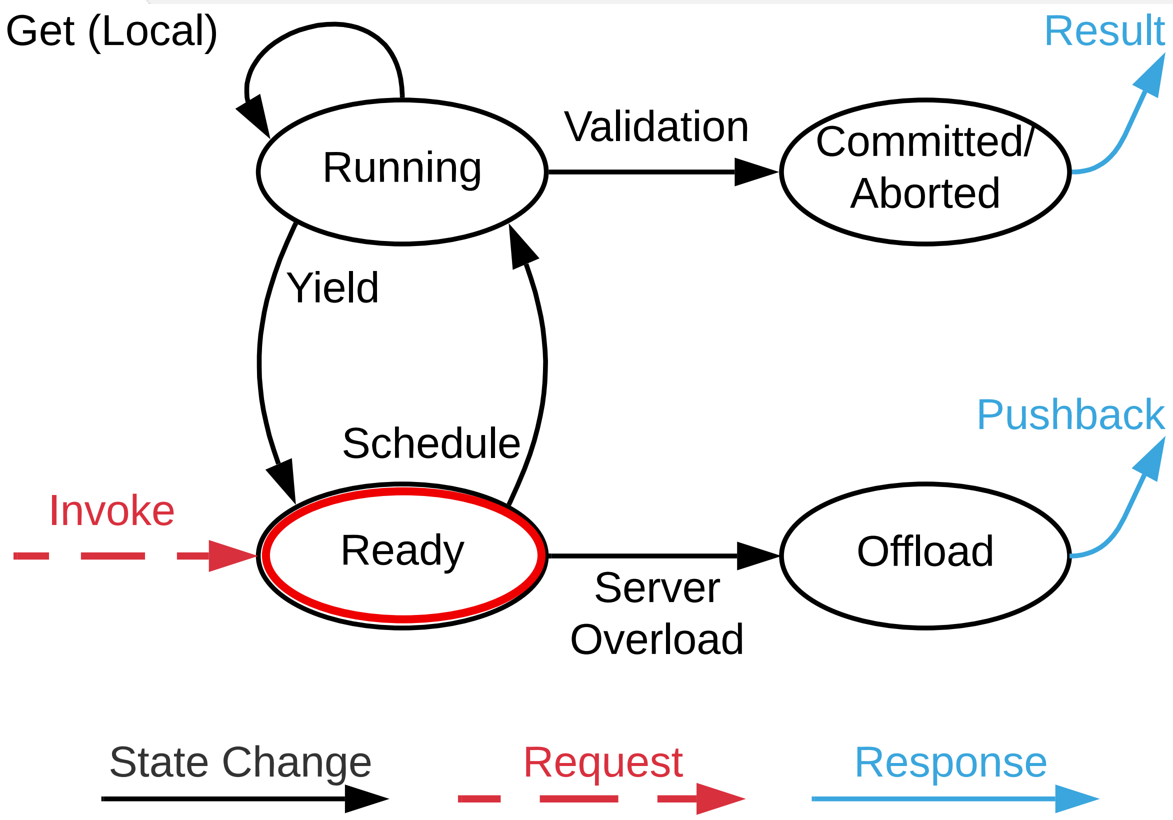


Running heavy compute at client creates room for remaining work

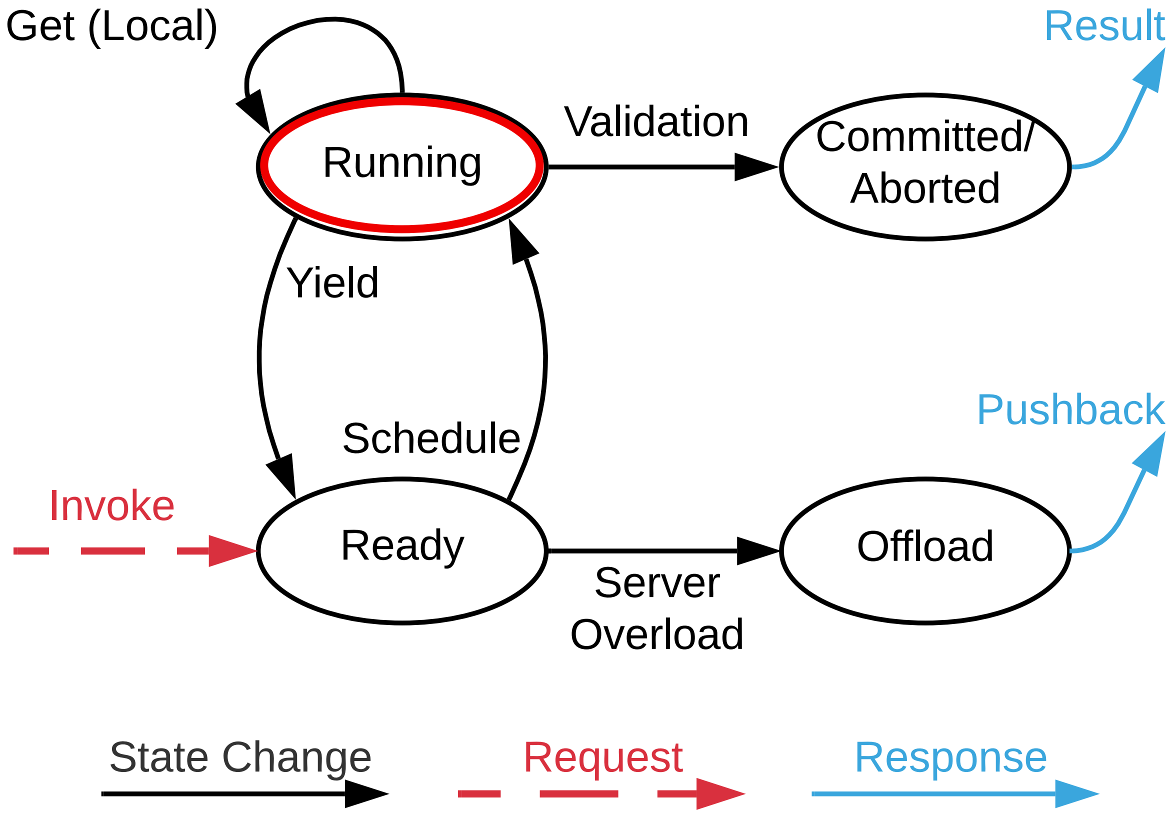
Adaptive Storage Function Placement (ASFP)

- Mechanisms
  - Server-side: Run Storage Functions, suspend, pushback to client
  - Client-side: Runtime, transparent remote data access
  - Consistency and concurrency control
- Policies
  - Invocation Profiling & Cost Modeling
  - Overload detection

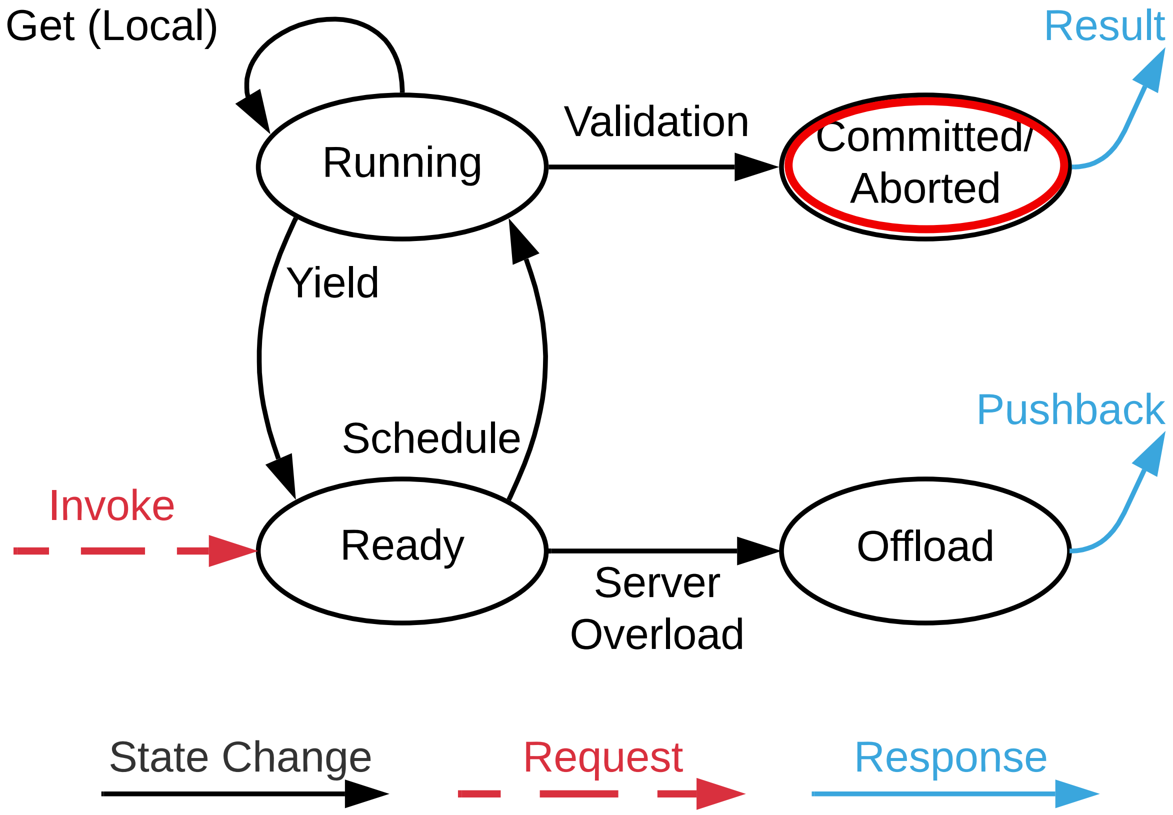
Server-side Storage Function Execution



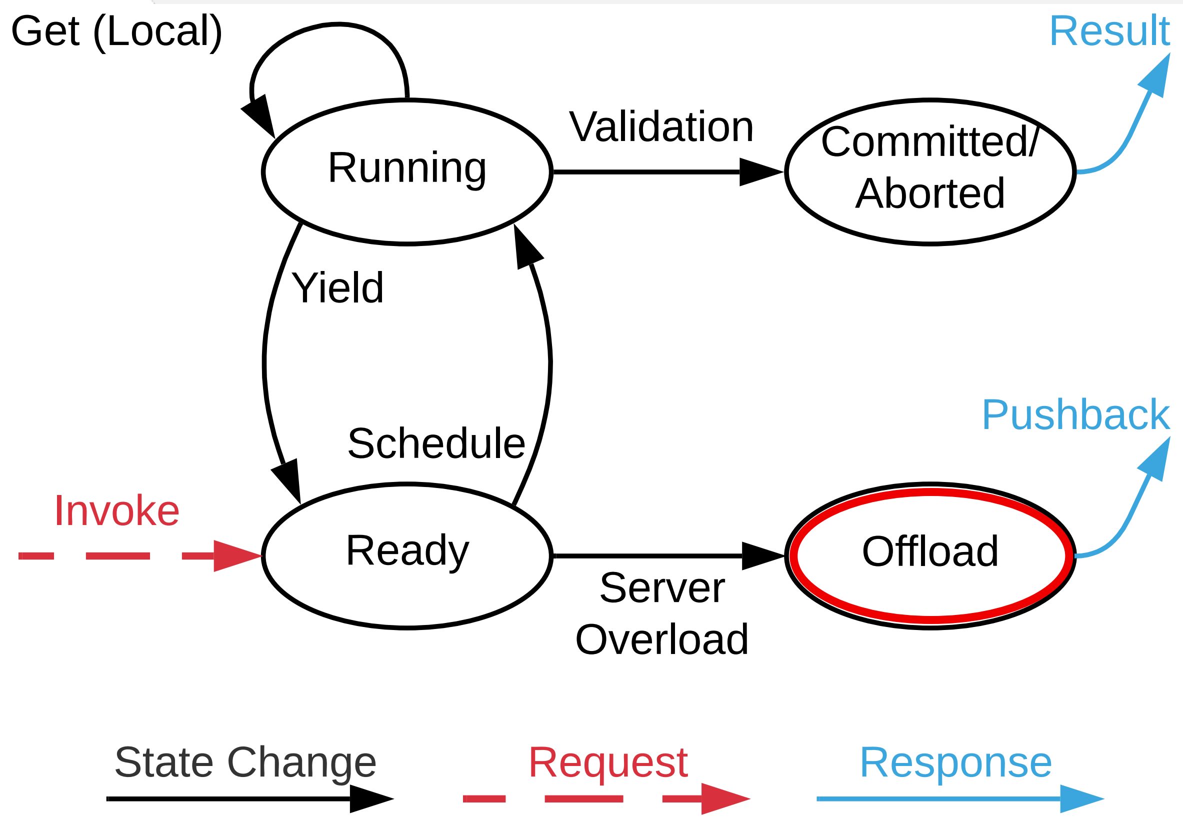
Server-side Storage Function Execution



Server-side Storage Function Execution



Server-side Storage Function Execution

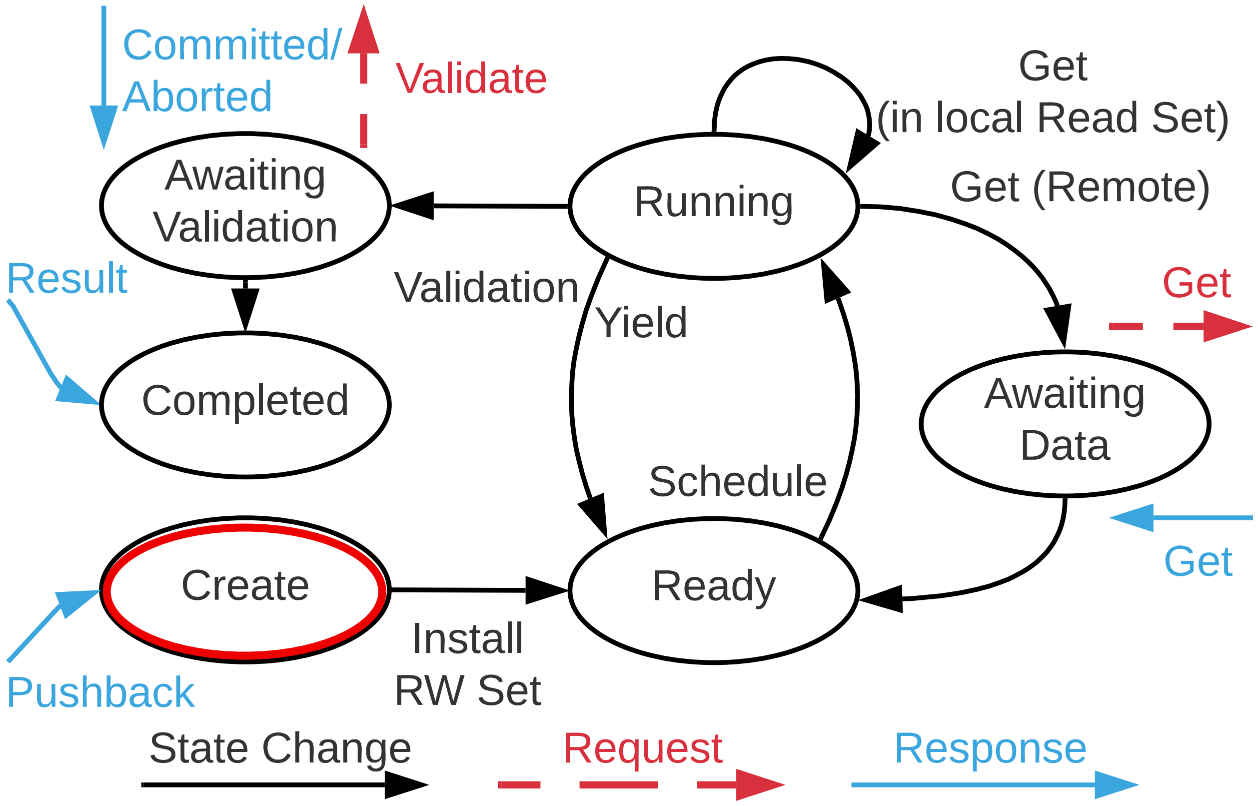


Consistency and Concurrency Control

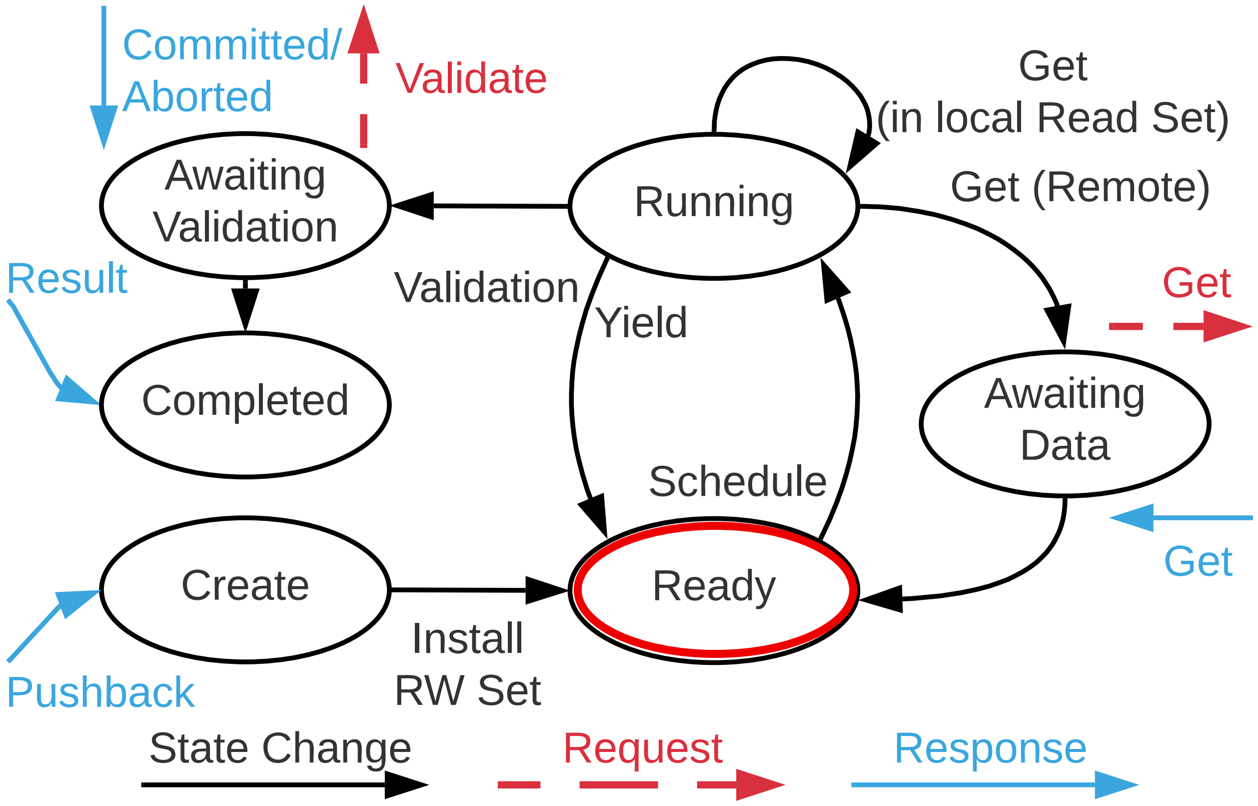
- **Problem:** `Invoke()` tasks run concurrently on server on each core and pushed-back invocations run in parallel to the server tasks
- **Solution:** Run invocations in strict serializable transactions
  - Use optimistic concurrency control (OCC)

- Read/Write set tracking is also used in pushback
  - Pushback invocation never generate work for Server
  - Server don't need to maintain any state for pushed-back invocations

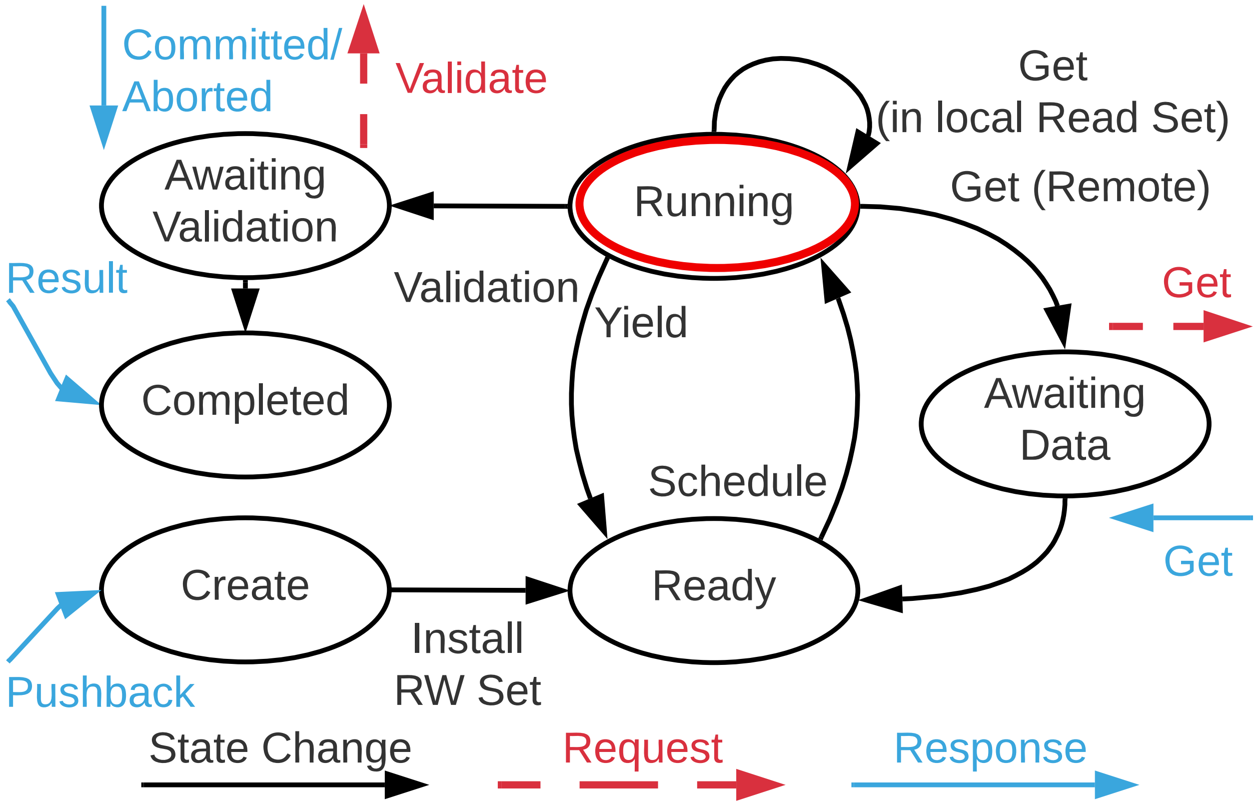
Client-side Execution for Pushed-back Invocations



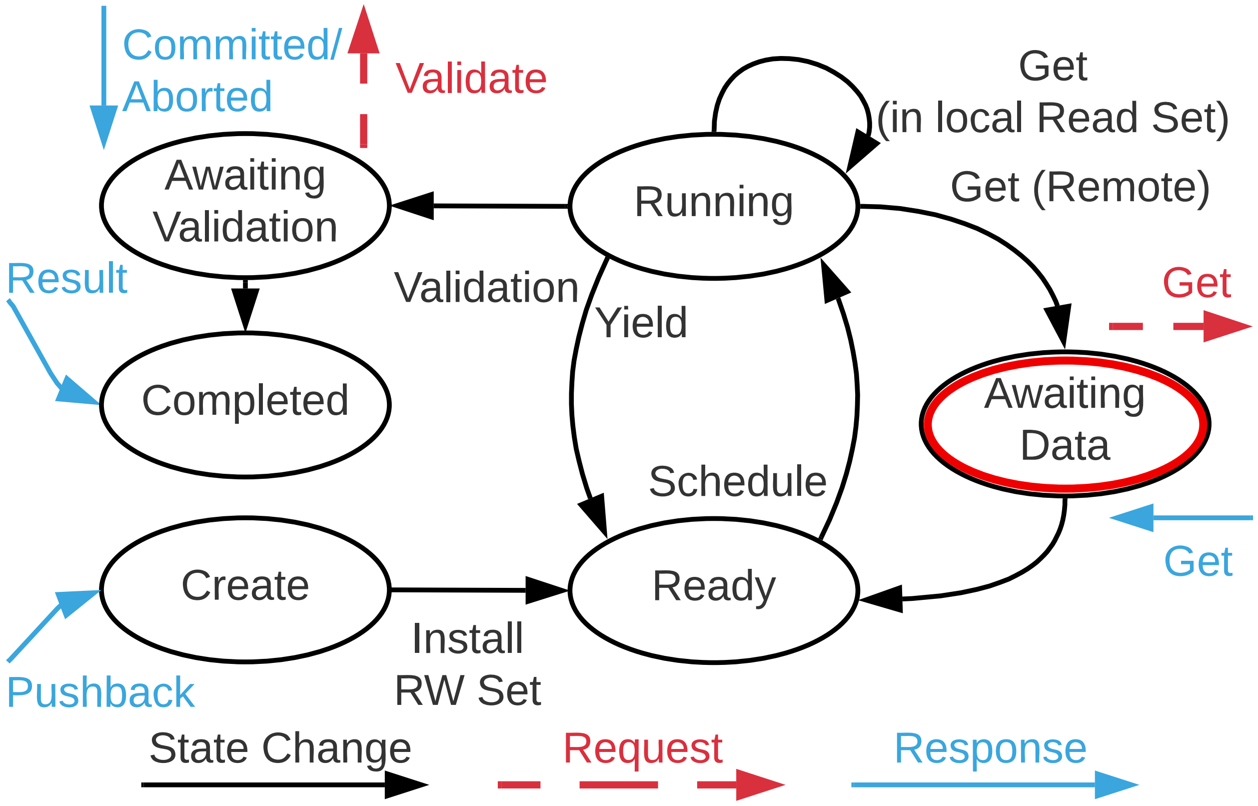
Client-side Execution for Pushed-back Invocations



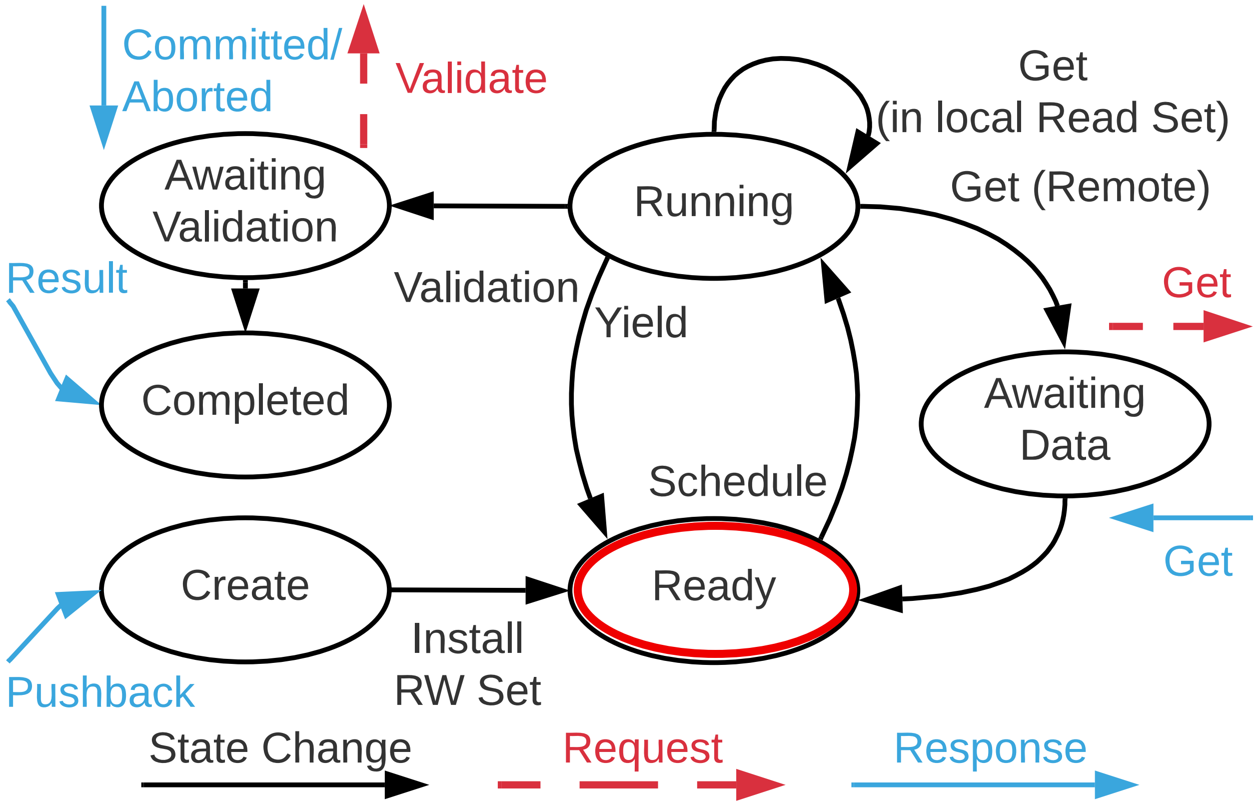
Client-side Execution for Pushed-back Invocations



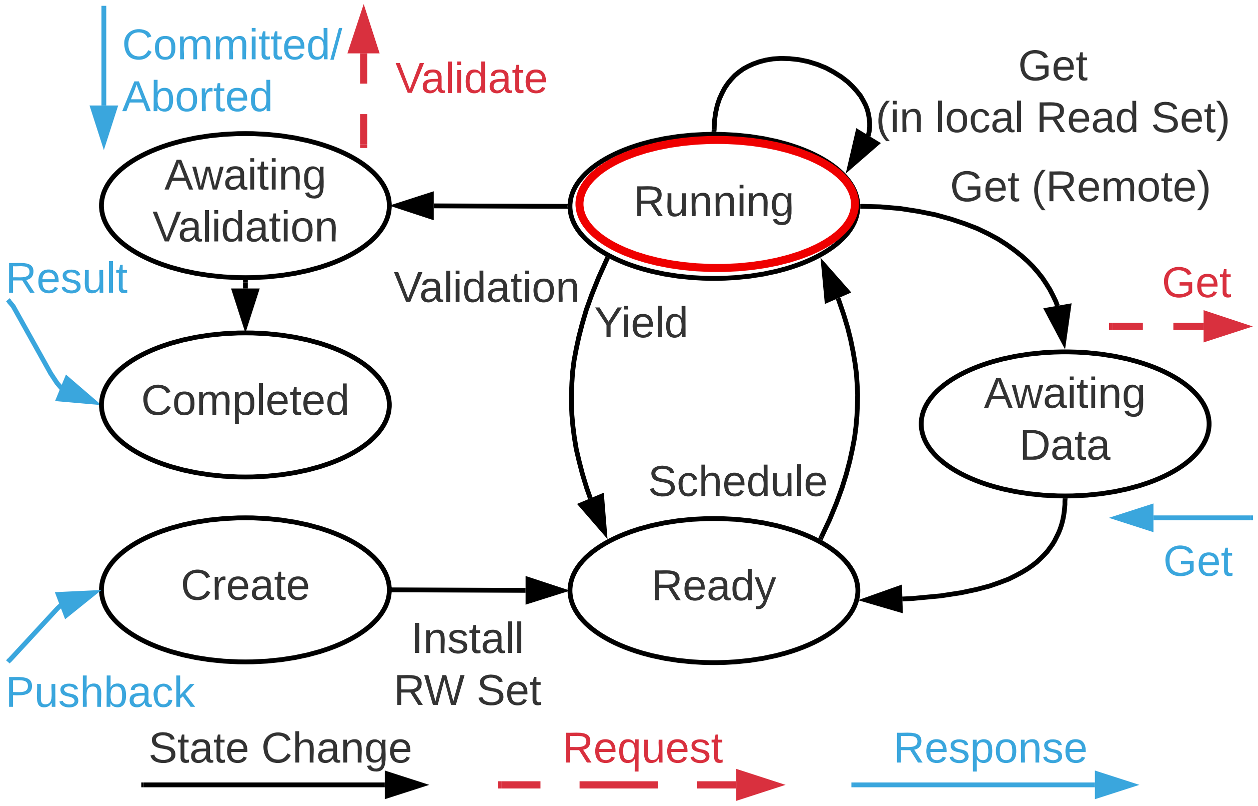
Client-side Execution for Pushed-back Invocations



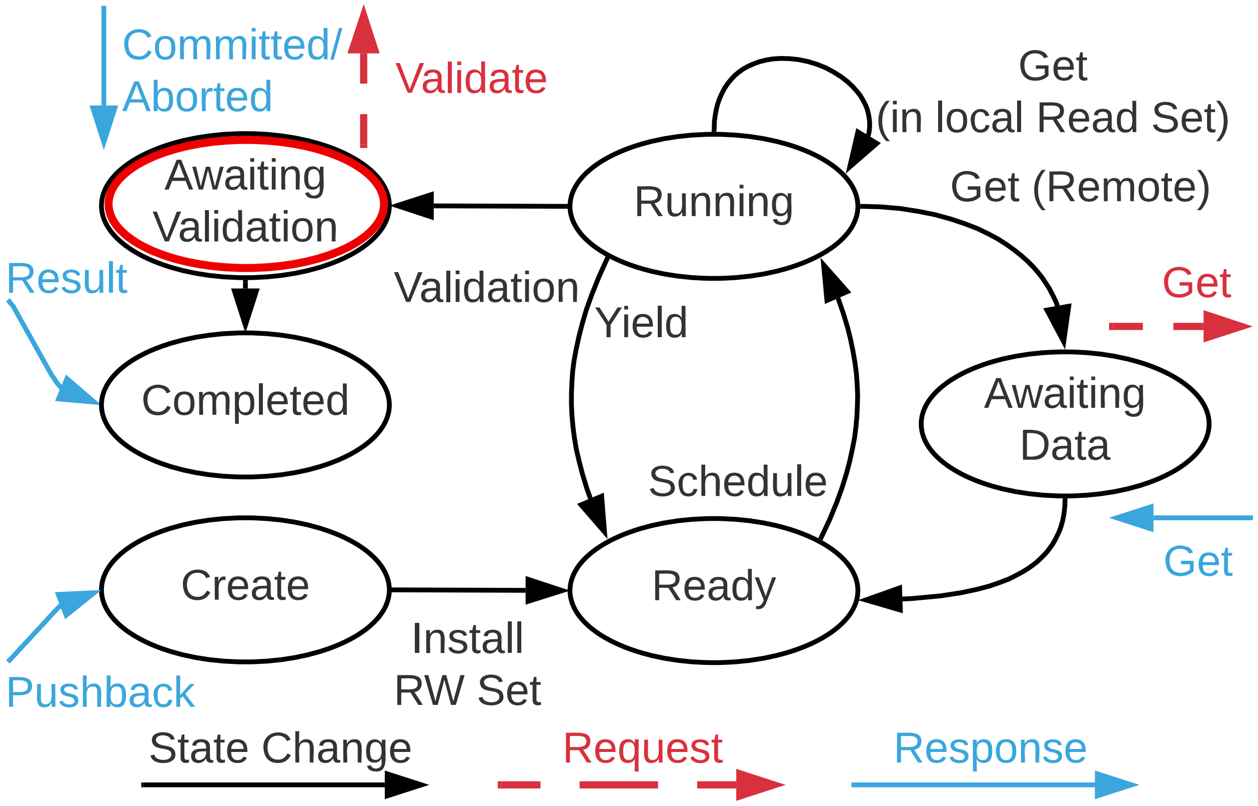
Client-side Execution for Pushed-back Invocations



Client-side Execution for Pushed-back Invocations



Client-side Execution for Pushed-back Invocations

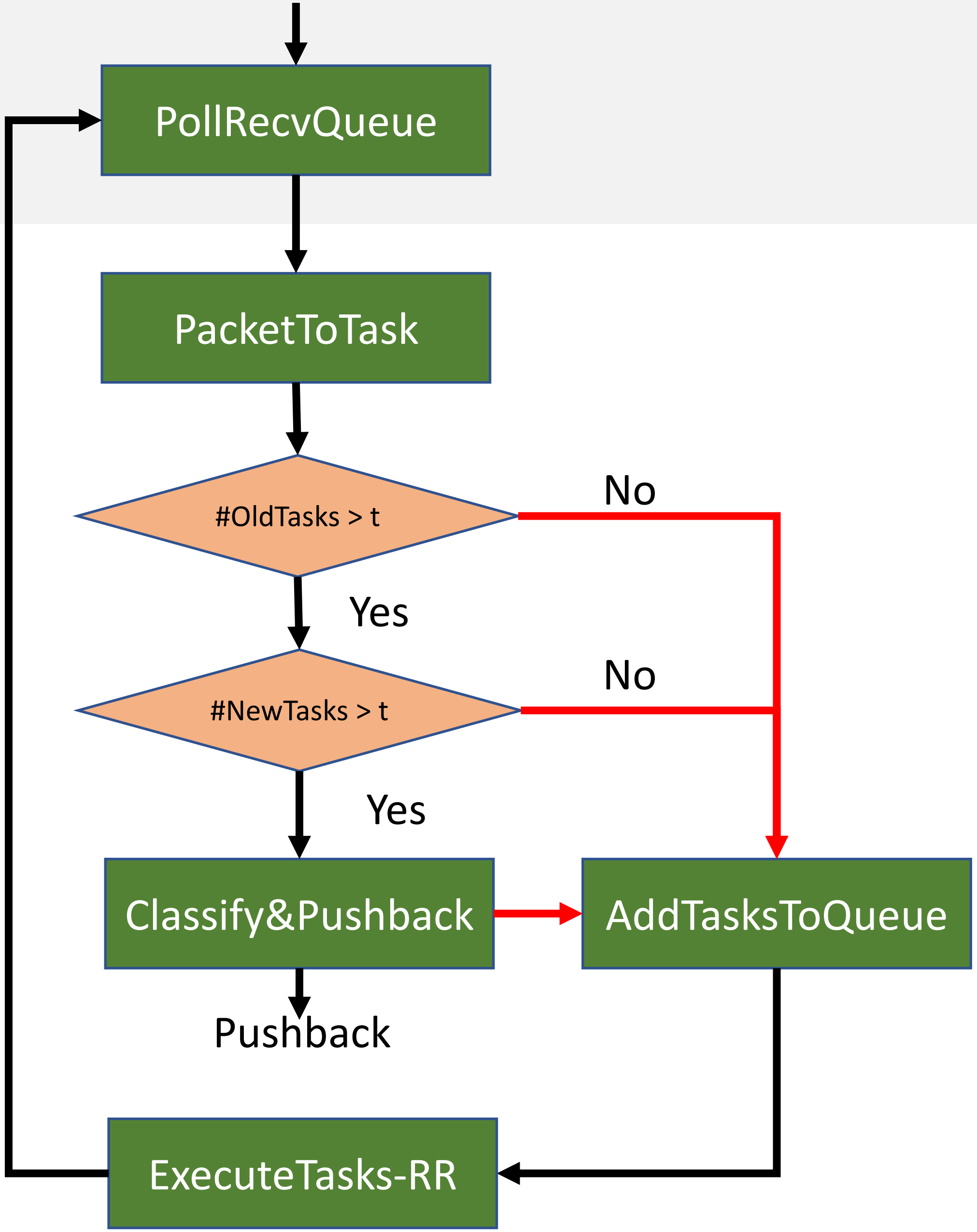


Adaptive Storage Function Placement (ASFP)

- Mechanism
  - Server-side: Storage Functions, suspend, move back to client
  - Client-side: Runtime, transparent remote data access
  - Consistency and Concurrency Control
- Policy
  - Server Overload Detection
  - Invocation Profiling and Classification

Server Overload Detection

- Always run the invocations on server, if underloaded
- Guarantees
  - Start pushback only when there are some old tasks and server receives even more tasks
  - Keep at least *t* tasks even after pushback, to avoid server idleness
  - Consider only `invoke()` tasks for overload detection



Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads

Invocation Profiling and Classification

- Profile each invocation for time spent in compute and data access
- Classify an invocation compute-bound if
  - Spent more time in compute than data access
  - Crossed a threshold $c > nD$
    - $c\ is\ amount\ of\ compute\ done\ by\ the\ invocation$
    - $n\ is\ the\ total\ number\ of\ data\ access\ till\ now$
    - $D\ is\ CPU\ cost\ to\ process\ one\ request$

Evaluation

GAINS AND COSTS

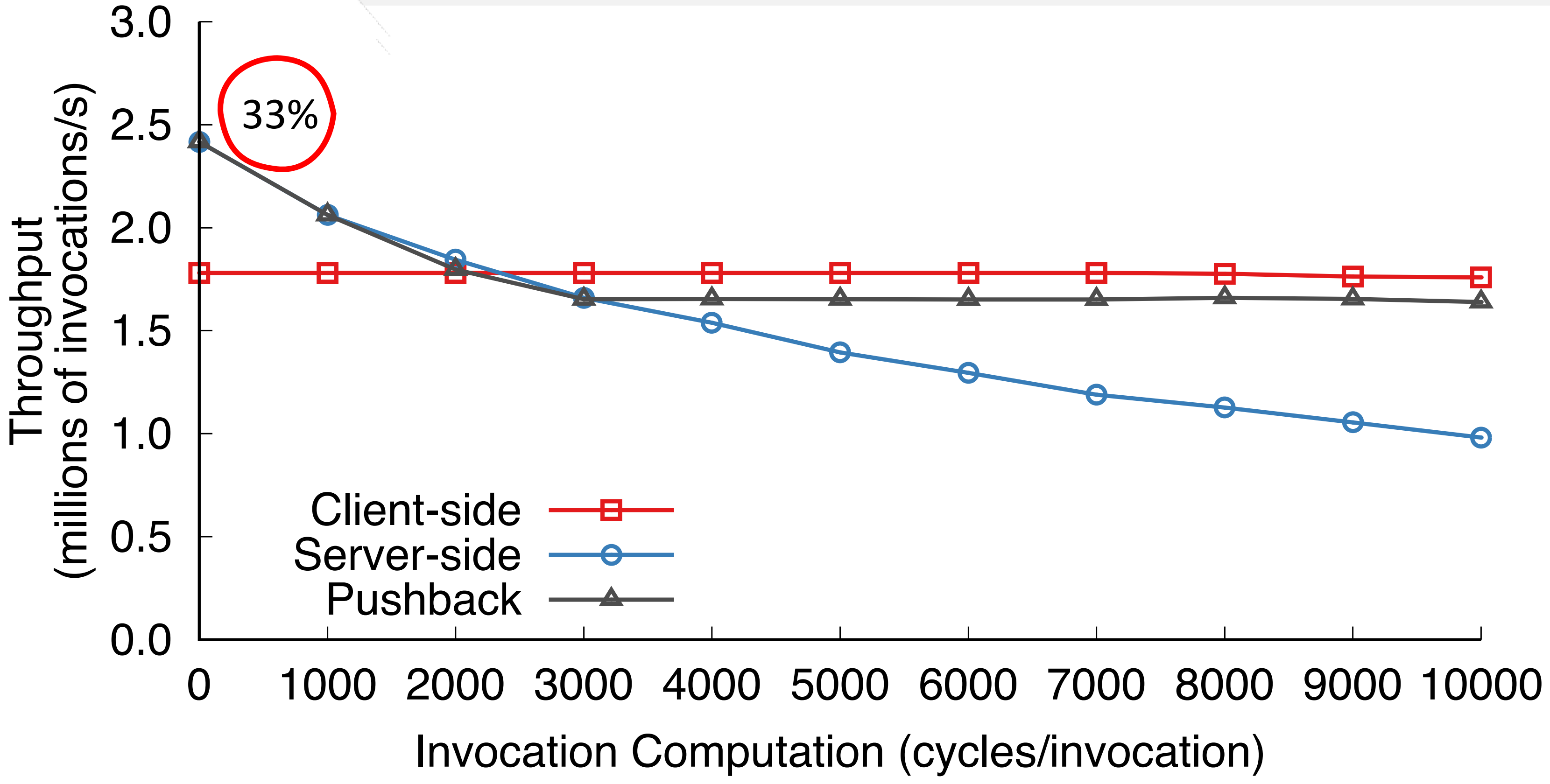
RW-SET EFFECT

APPLICATION MIX

Experimental Setup

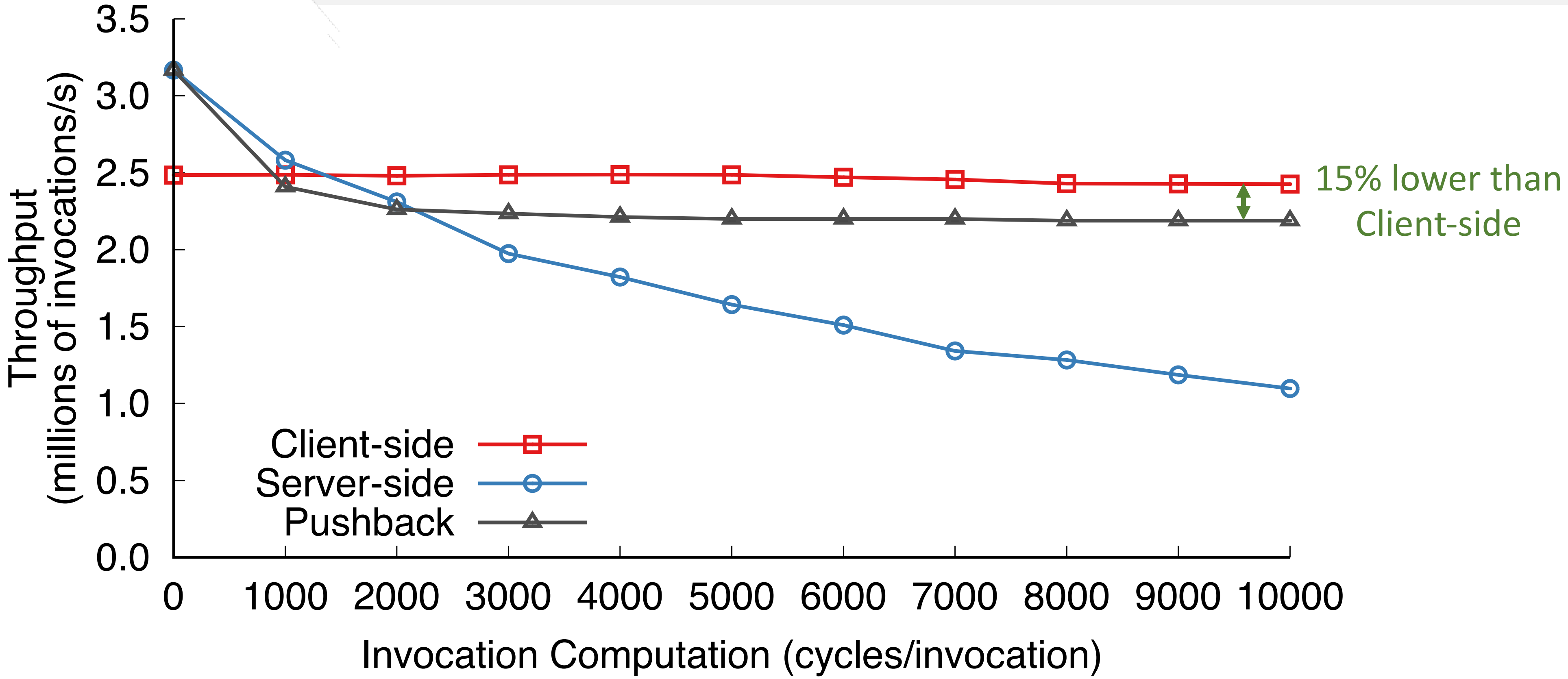
- One Server and Four Client
  - CPU - Ten-core Intel E5-2640v4 at 2.4 GHz
  - RAM - 64GB Memory (4x 16 GB DDR4-2400 DIMMs)
  - NIC - Mellanox CX-4, 25 Gbps Ethernet
- 15GB Read-write set as 120M Records, 30B key and 100B value

Does ASFP improve server throughput?



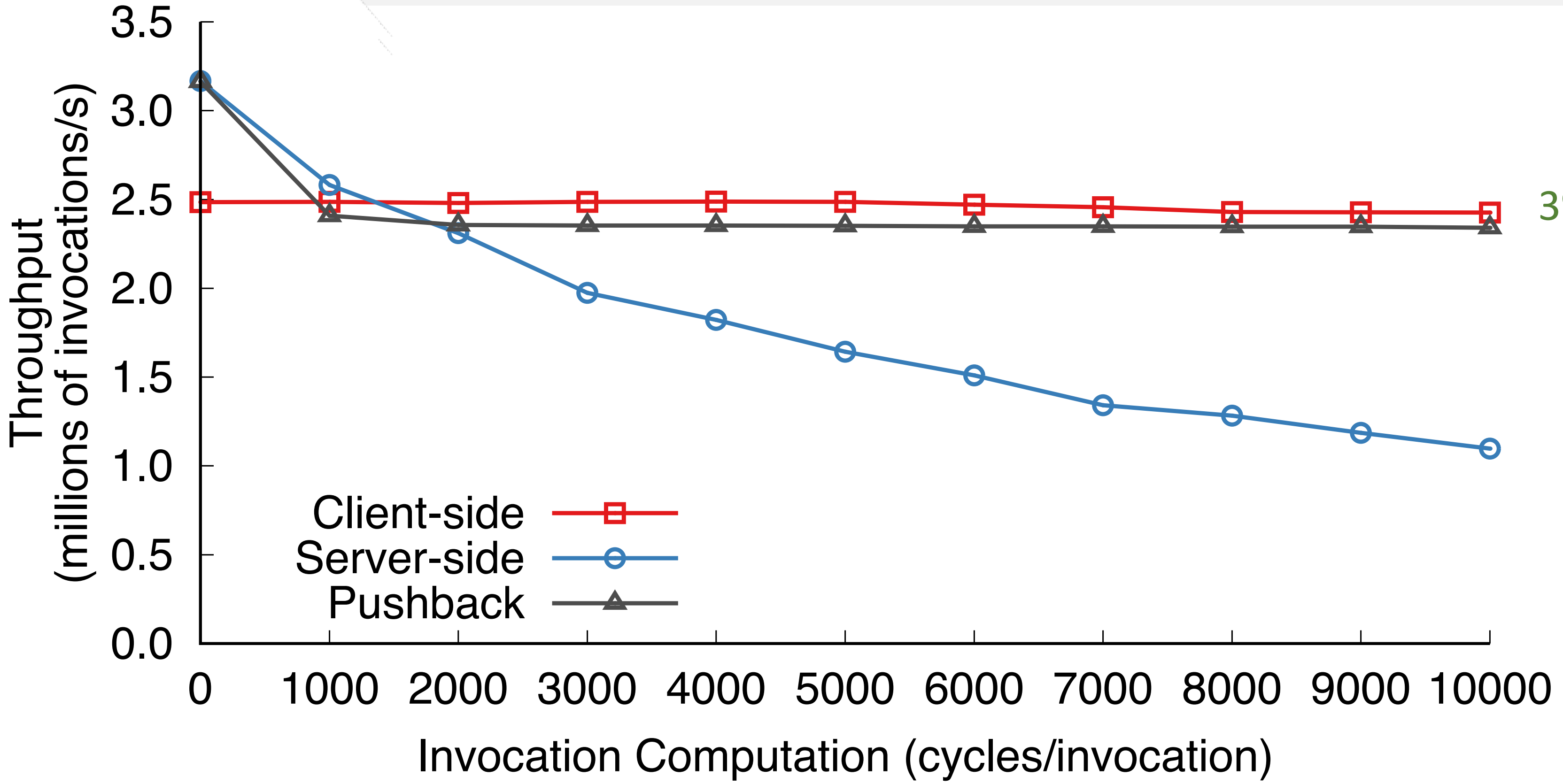
3 data-accesses per invocation

What is the cost of using ASFP?



2 data-accesses per invocation

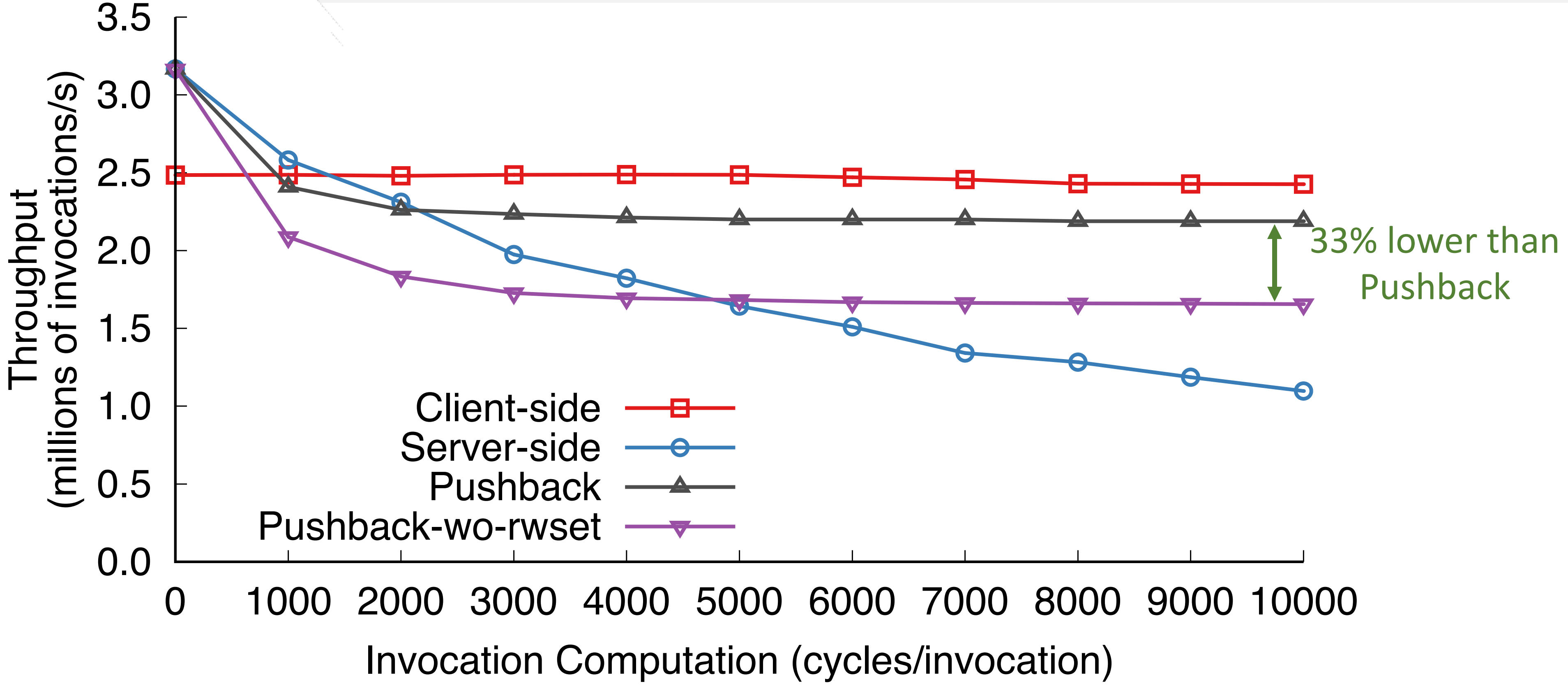
What is the cost of using ASFP?



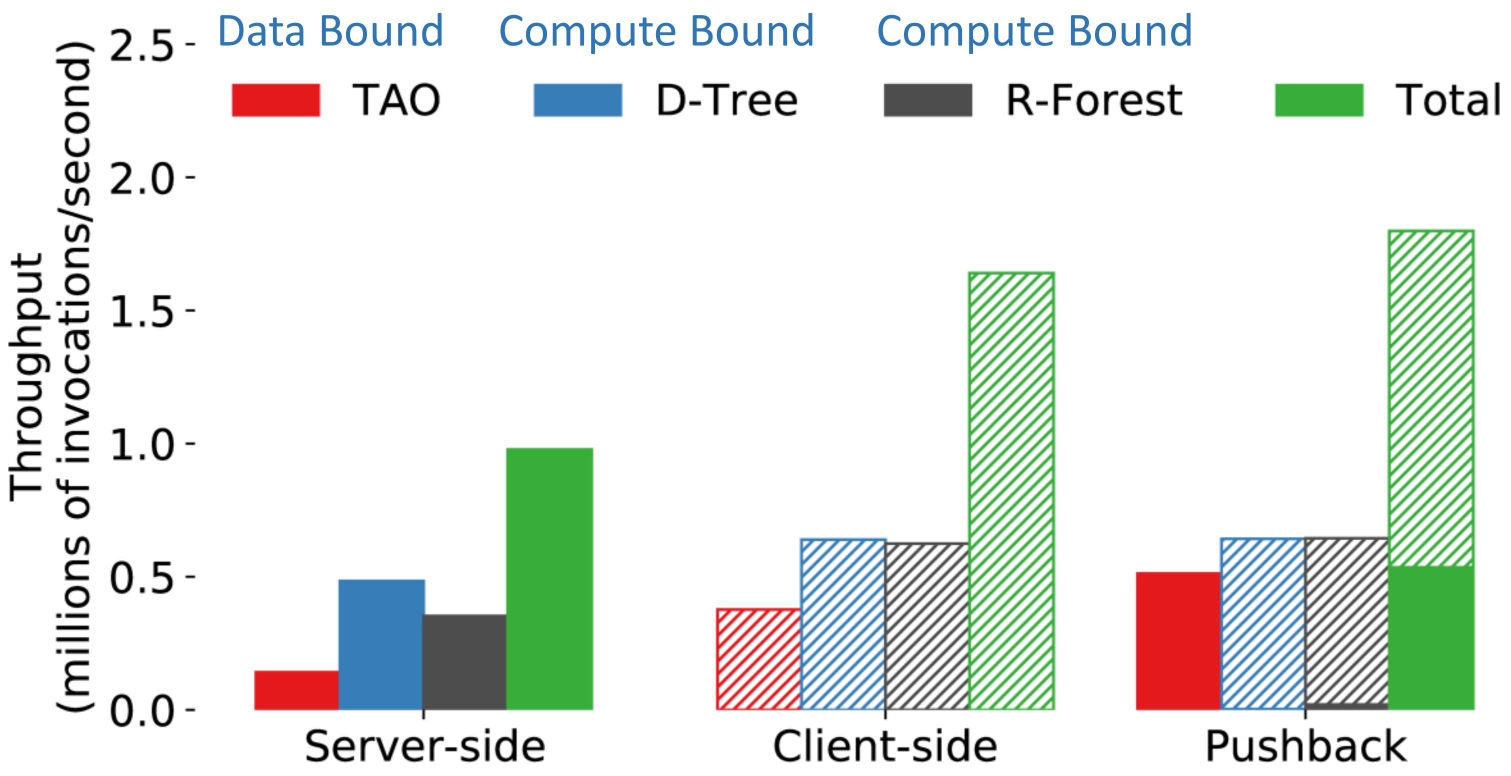
3% lower than Client-side

Aggressive overload detection

How do ASFP and OCC interact?

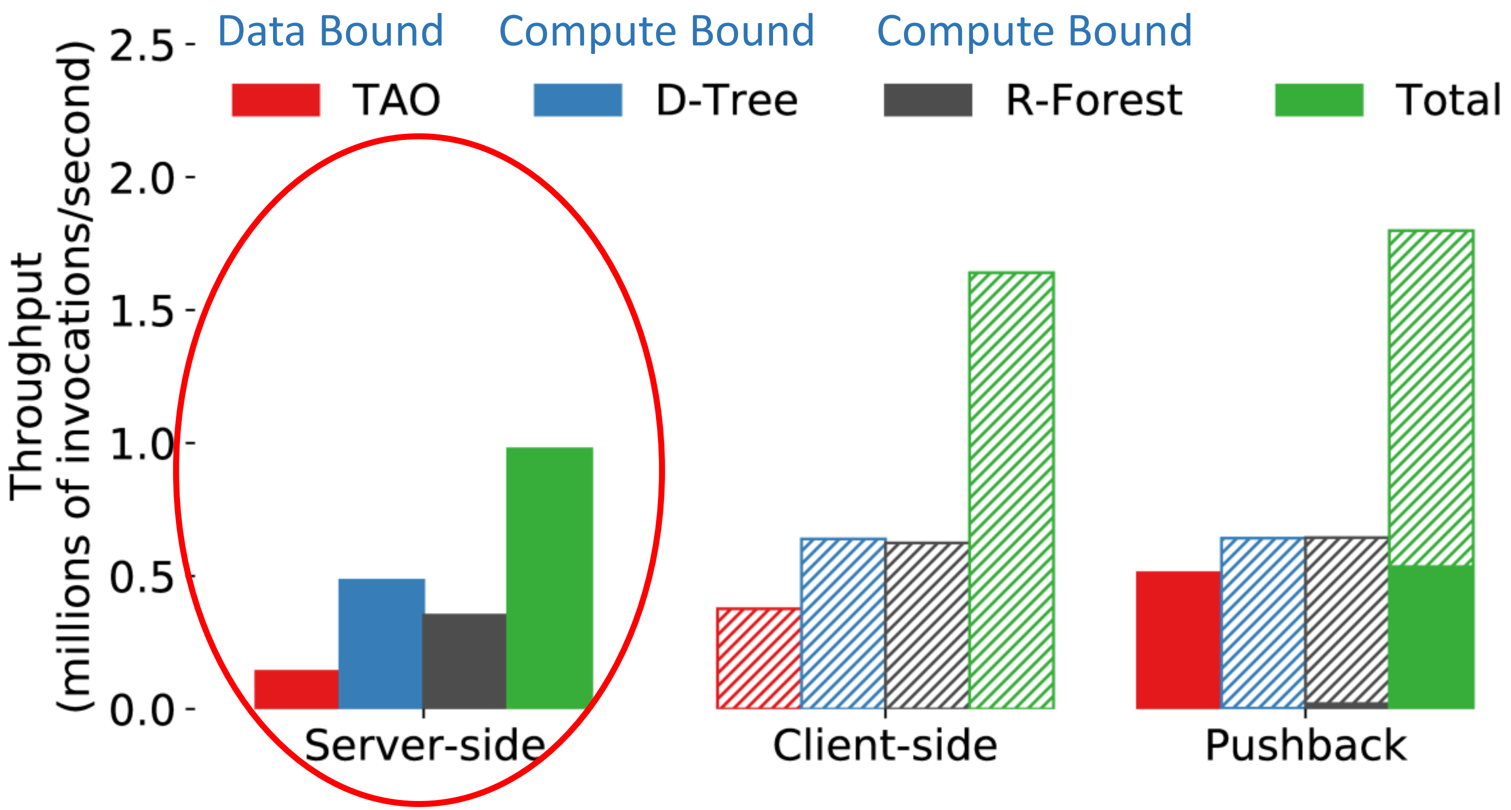


Does ASFP improve throughput for an Application Mix?



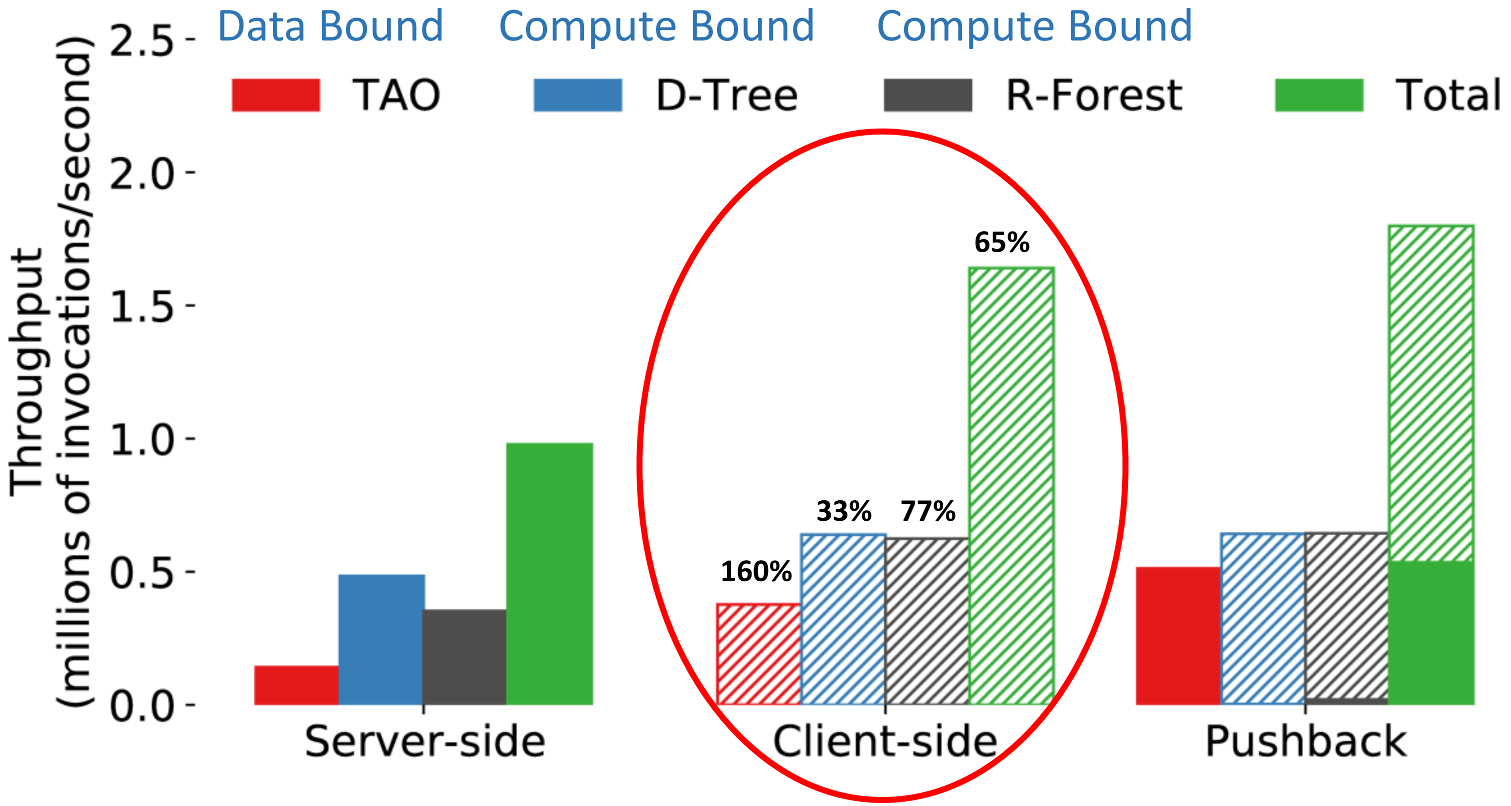
Solid: Run Server-side, Hashed: Run Client-side

Does ASFP improve throughput for an Application Mix?



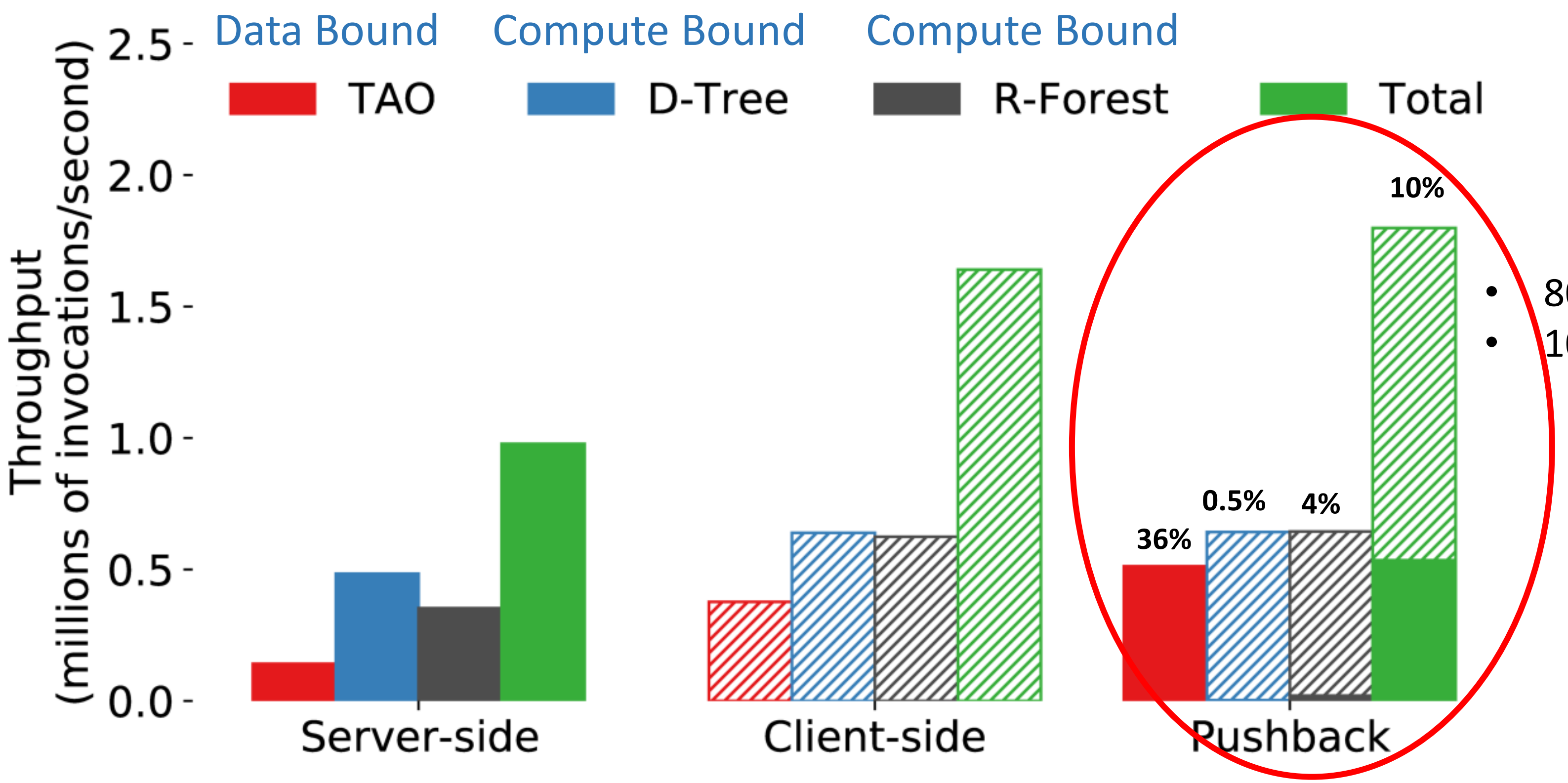
More room on server to respond to more get/puts

Does ASFP improve throughput for an Application Mix?



More room on server to respond to more get/puts

Does ASFP improve throughput for an Application Mix?



- 80% higher than Server-side
- 10% higher than Client-side

TAO ↑ by avoiding data movement; Pushback makes room for TAO

Related Work

- Storage Procedures, UDFs
  - SQL - Poor fit for specialized computation
  - Redis – Extension provided at server start time
  - Splinter- build on top of it

- Offloading and code migration in mobile and edge computing
  - MAUI – different timescales and use-cases

- Thread and Process Migration
  - Sprite, Condor – slow and unsuitable for µs scale

Conclusion

- **Kernel-bypass** key-value stores offer < **10µs** latency, **Mops** throughput
  - Fast because they are just dumb
  - Inefficient – Data movement, client stalls
- Run application logic on the server?
  - Storage server can become bottleneck, effects propagates back to clients
- Adaptively place the invocations to avoid bottlenecks
  - Up to 42% gain for low-compute invocations (vs client-side)
  - Comparable performance for high-compute invocation(vs client-side)